A Data Oriented Approach for Real-Time Systems

Tanguy Le Berre, Philippe Mauran, Gérard Padiou, Philippe Quéinnec

In Proc. of the 17th International Conference on Real-Time and Network Systems
RTNS'2009, Paris, ECE, 26-27 October, 2009

A Data Oriented Approach for Real-Time Systems

Tanguy Le Berre, Philippe Mauran, Gérard Padiou, Philippe Quéinnec
Université de Toulouse - IRIT
2, rue Charles Camichel
31000 TOULOUSE, FRANCE
{tleberre,mauran,padiou,queinnec}@enseeiht.fr

Abstract

Distributed real-time systems often have to maintain the temporal validity of data. In this paper we present a modelling framework centered on data where a so-called observation relation represents and abstracts the interactions between variables. An observation is a relation between variables, an image and its sources, where the image values depend on past values of the sources. The system architecture is seen as a set of observation relations describing the flow of values between variables. The observation relations are parametrized with timed constraints that limit the time shift between the variables and specify the availability of timely sound values.

At this level of abstraction, the designer gives a specification of the system based on timed properties about the timeline of data such as their freshness, latency etc. We proceed to an analysis of the feasibility of such a specification and we formally analyze the correctness of an implementation with respect to a specification.

In order to prove the feasibility of an observation-based model, we build a finite state transition system which is bi-similar to the specification. The existence of an infinite execution in this system proves the feasibility of the specification. Possible implementations are described as a set of interacting components which control the flow of values in the system. A finite system is built to prove the correctness of the implementation by model checking.

1 Introduction

We propose a framework to specify and analyze the timed properties of distributed real-time systems. The architecture of a system is not described as a set of communicating tasks. It is rather described as a set of related variables and links between the values of the variables. The timed requirements of the system are expressed on these links and state that the values of a variable that are available in the system must be ”timely valid”. A value is valid if it based on values of other variables that are consistent with the environment and the user's requirements. The goal is to express the timed requirements regardless of the task and communication protocols We then check that these requirements are satisfied by the implementation.

This paper presents the formal definitions used to build and analyze our framework. This modelling framework is illustrated by a simple example, an automatic cruise control system. We first describe the underlying formal system. We then introduce the observation relation that is used to describe the architecture of the system as a set of links between the values of the variables. Based on these links, a set of timed properties is defined to specify the timed requirements. A system is specified by the architecture and the timed properties. We explain how the feasibility of the system is proven by using the specification to build a bi-similar finite state transition system which is then explored to search for possible executions. Finally, we show how to model an implementation and check its correctness with respect to the specification. Here also a dedicated state transition system is built.

2 Related Works

A typical approach to real-time systems is the specification of properties as characteristics of the tasks. A scheduling analysis is then performed to check the satisfaction of these properties. In the case of distributed systems, the scheduling analysis takes into account the properties of the communication protocol as in [9].

We depart from such an analysis by expressing the properties as state based properties on the system variables. The properties are not expressed on the tasks and so do not relate system events.

Most works where the properties are specified on the data belong to the field of databases. For example, the variable semantics and their timed validity domains are used in [10] to optimize database transaction scheduling. Our work stands at a higher level as we propose to give an abstract description of the system in terms of of data relations. Another work analyzes the propagation of value in real-time database and their timed correctness [3]. But they only give results as a synchronized set of period tasks. In [8], the authors define derived objects that are computed from a set of objects. The age of an object is defined by the ages of the objects used to compute it. Their goal is to find

a scheduling of a set of periodic preemptable transactions to maintain mutual consistency. They want to limit the dispersion of the ages of the set of objects used to compute a derived object. In this paper we want to check other timed properties such as the freshness of the used objects.

Similar works specify systems using temporal logic. In [2], OCL constraints are used to define the validity domain of variables. A variation of TCTL is used to check the system synchronization and prevent a value from being used out of its validity domain. This work also defines timed constraints on the relations between application variables, but these relations are defined using events such as message sending whereas our definitions are based on the history of the values of the variables.

In [7], an Allen linear temporal logic is proposed to define constraints between intervals during which state variables remain stable. As in our approach, it uses an abstraction of the data timelines in terms of stability intervals. However in [7] the constraints do not relate to real-time.

3 An Introducing Example

We introduce a system example used to illustrate our framework. This system is a simplified automatic car cruise control system. The goal of such a system is to control a vehicle by maintaining a steady speed chosen by the driver. The vehicle speed is controlled through a throttle actuator. A sensor is used to compute the vehicle's current speed and based on this speed and the speed chosen by the driver, the input of the throttle actuator is computed by the control system. The architecture of this simplified system is illustrated Figure 1. This system is a distributed system where the components communicate through a bus.

This system reacts to its environment. The evolution of the vehicle speed implies that each value submitted to the throttle actuator has a bound validity domain. Thus, there are timed requirements on the speed at which the system reacts. We informally give the timed requirements and properties on data and relations between data in such a system:

- the current speed is computed based on the wheel turns. So, a minimum duration between each computation is required to give a relevant speed;
- but the speed must be updated often enough to be consistent with reality;
- there is a minimum time between two updates of the desired speed ;
- due to the bus properties, there is a minimum communication time between the different components;
- the throttle actuator value must be consistent with the current value of the vehicle current speed and the desired speed.

We explain how our approach allows to formally define this system and its real-time properties. Each component uses and/or produces data. We use a relation called observation to specify the dependencies between variables.

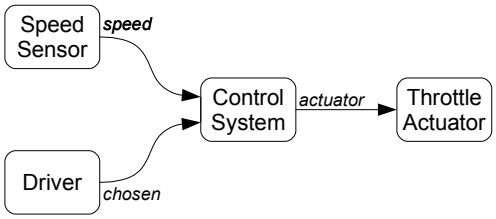

Figure 1. Cruise Control System

4 Formal Background

We give here the formal context and the definition of the properties used to define the observation relation and system timed properties.

4.1 State Transition Systems

Models used in this paper are based on state transition systems. A *transition system* S is a couple $(\Sigma, \rightarrow)$ where Σ is a set of *states* and $\rightarrow$ is a *transition relation*, i.e. a predicate on pair of states. A *state* is an assignment of values to variables. A *step* is a pair of states which satisfy the transition relation. An *execution* σ is any infinite sequence of states $\sigma_0\sigma_1 \ldots \sigma_i \ldots$ such that two consecutive states form a step. We note $\sigma_i \rightarrow \sigma_{i+1}$ the step between the two consecutive states σ_i and σ_{i+1}.

The system properties are expressed as *temporal predicates*. A *temporal predicate* is a predicate on executions; we note $\sigma \models P$ when an execution σ satisfies the predicate P. Such a predicate is written in linear temporal logic. A *state expression* e (in short, an expression) is a formula on variables; the value of e in a state σ_i is noted $e.\sigma_i$. The sequence of values taken by e during an execution σ is noted $e.\sigma$. A *state predicate* is a boolean-valued expression on states.

4.2 Introducing Time

We consider real-time properties of the system data. To distinguish them from (logical) temporal properties, such properties are called *timed* properties. Time is integrated in our transition system in a simple way, as described in [1]. Time is represented by a variable T taking values in an infinite totally ordered set, such as $\mathbb{N}$ or $\mathbb{R}^+$. The time domain is called $\mathbb{T}$. T is an increasing and unbound variable. There is no condition on the density of time and moreover, it makes no difference whether time is continuous or discrete (discussion in [6]). However, as an execution is a sequence of states, the actual sequence of values taken by T during a given execution is necessarily discrete. Note that we explicitly refer to the variable T to study time.

An execution can be seen as a sequence of snapshots of the system, each taken at some instant of time specified by the value of T. We require that "enough" snapshots are performed to catch each computation step. It means that no variable can have different values at the same time and so in the same snapshot. Any change in the system implies time passing.

Definition 1 *Separation. An execution σ is separated iff for any variable x:*

$$\forall\, i,j : T.\sigma_i = T.\sigma_j \Rightarrow x.\sigma_i = x.\sigma_j$$

In the following, we consider only separated executions. This allows to timestamp updates of variables.

4.3 Clocks

Let us consider a totally ordered set of values $\mathcal{D}$, such as $\mathbb{N}$ or $\mathbb{R}^+$. A clock is a (sub-)approximation of a sequence of $\mathcal{D}$ values. We note $[X \rightarrow Y]$ the set of functions with domain X and range contained by Y.

Definition 2 *A clock c is a function in $[\mathcal{D} \rightarrow \mathcal{D}]$ such that:*

- *it never outgrows its argument value:* $\forall\, t \in \mathcal{D} : c(t) \leq t$
- *it is monotonously increasing:* $\forall\, t,t' \in \mathcal{D} : t < t' \Rightarrow c(t) \leq c(t')$
- *it is lively:* $\forall\, t \in \mathcal{D} : \exists\, t' \in \mathcal{D} : c(t') > c(t)$

The predicate $clock(c)$ is true if the function c is a clock.

In the following, clocks are used to characterize the timed behavior of variables. They are defined on the indices of the sequence of states, to express a logical precedence.

4.4 Data Timeline

In order to state properties on the timed behavior of a variable x, we have to be able to characterise its timeline. We introduce a variable that refers to the last time this variable was updated. These are called the update instants $\hat{x}$. The goal is to capture the instant when the current value of x appeared, e.g. the beginning of the current occurrence. This referential can be either explicit or implicit. In the explicit case, the developer is responsible for giving its own variable $\hat{x}$. For example, when a variable is updated in a periodic way. In the implicit case, a formal definition of $\hat{x}$ is given based on the history of the values taken by x.

Definition 3 *For a separated execution σ and a variable x, the update instants of x is:*

$$\forall\, i : \hat{x}.\sigma_i = T.\sigma_{min\{j|\forall k \in [j..i]:\, x.\sigma_i = x.\sigma_k\}}$$

The timeline $\hat{x}$ is built from the history of x values. For a variable x, the update instant of x is defined as the value taken by the time T at the earliest state when the current value appeared and continuously remained unchanged up to the current state.

When x is updated and its value changes then the value of $\hat{x}$ is also updated. Conversely, if $\hat{x}$ changes then the value of x is modified. We consider in this paper that whenever a variable is updated, it is with a new value so that the update instants are equivalent to the modification instants.

The variable d_x is also defined to stand for the duration the current value of x is continuously kept.

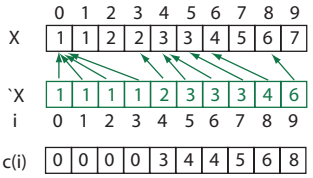

Figure 2. The Observation Relation

Definition 4 *For a separated execution σ and a variable x, the variable d_x is defined by:*

$$\forall\, i : d_x.\sigma_i = T.\sigma_{min\{j|\forall k \in [i..j[:x.\sigma_i = x.\sigma_k\, \wedge\, x.\sigma_i \neq x.\sigma_j\}} - \hat{x}.\sigma_i$$

These two variables give the timed characteristics of the current value of the variable.

5 Modelling the Data Flow

5.1 The Observation Relation

To give properties on the relations binding variables, we define an operator, the observation relation, on state transition systems as in [4]. The observation relation is used to abstract the dependency between values taken by different variables.

More precisely the observation relation binds two variables, the source x and its image y, and denotes that the history of the variable y is a sub-history of the variable x. The relation is defined by one couple $\langle source, image \rangle$ and the existence of at least a clock defining for each state which of the previous values of the source is taken by the image. This clock is used to define the time shift introduced by the observation. Figure 2 shows an example of an observation relation. The definition is:

Definition 5 *The variable y is an observation of the variable x in an execution σ: $\sigma \vDash y \prec x$ iff:*

$$\exists\, c \in [\mathbb{N} \rightarrow \mathbb{N}] : clock(c) \wedge\ \forall\, i : y.\sigma_i = x.\sigma_{c(i)}$$

This relation is used to abstract the communication in a distributed system. We extend this definition to a relation binding an image to a set of variables and a function.

Definition 6 *Given a function f and a set of variables $X = \{x_i | i \in [1..n]\}$, the variable y is an observation of the expression $f(X)$ in execution σ: $\sigma \vDash y \prec f(X)$ iff:*

$$\exists\, c \in [\mathbb{N} \rightarrow \mathbb{N}] : clock(c) \wedge$$
$$\forall\, i : y.\sigma_i = f(x_1.\sigma_{c(i)}, x_2.\sigma_{c(i)}, \ldots, x_n.\sigma_{c(i)})$$

In this case, all values of the inputs (X) are read at the same time, implying a synchronous behavior. Then the inputs are at the same node or the different nodes have to be perfectly synchronized. If they are not, additional observation relations are added to model the communication and the copy of the input to the computation node. With this definition, the basic observation is just a special case where $f = Identity$ and $card(X) = 1$.

Thus the observation can be used as an abstraction of communication in a distributed system as well as an abstraction of a computation:

- communications:
  $`speed \prec speed$
  $`chosen \prec chosen$
  $`actuator \prec actuator$
- computation:
  $actuator \prec control(`speed, `chosen)$

Figure 3. System Architecture

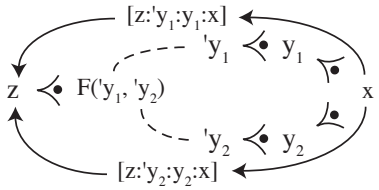

Figure 4. Path between Variables

- Communication consists in transferring the value of a local variable to a remote one. Communication time and lack of synchronization create a lag between the source and the image, which is modelled by $distant \prec local$.
- In state transition systems, a computation $f(X)$ is instantaneously computed. By writing $y \prec f(X)$, we model the fact that the computation takes time and that the value of y is based on the value of the inputs (X) at the beginning of the computation.

5.2 Example

We use the observation relations to describe the architecture of the example (see Figure 3). The distribution of the system is defined by the image variables $`speed, `chosen$ and $`actuator$. These variables are copies of the values of the variable $speed, chosen$ and $actuator$ sent through the communication bus.

The computation of the value of the $actuator$ variable is based on the values of the variables $`speed$ and $`chosen$. A $control$ function defines the computation of the variable $actuator$. This function is used in an observation relation binding the $actuator$ variable with the copies of the current speed and the chosen speed.

5.3 Path between Variables

Even if two variables are not directly related by any observation relation, they can be related by a set of observations. In the example, the value of $`actuator$ indirectly depends on the values of the variable $speed$. We want to be able to describe such an indirect dependency.

A set of observation relations defines an oriented graph where each variable is a node and observations are the edges that link the sources to the images. Given a set of observations, two variables are linked if there is a path between the nodes of these variables. Such a path represents the propagation of variable values through the system.

When none of the observations model a computation, there always exists a unique observation path between a given source and an image. But several observation paths can appear when a computation involves several source arguments. In figure 4, $F(`y_1, `y_2)$ has two images as input data, so two distinct observation paths have to be separately studied to verify time properties attached to the pair (z, x). Therefore, we define an observation path as a distinguishable sequence of variables. For example, the two paths between z and x are defined by the sequences $[z : `y_1 : y_1 : x]$ and $[z : `y_2 : y_2 : x]$.

The timed properties of the system are defined as properties on the propagation time of the values between two nodes. They express the time shifts that are introduced by the system architecture.

For each observation relation, the time shift is defined by the observation clock. The time shift along a path is defined by the composition of the observation clocks.

Definition 7 *Given the set of observation relations Obs that defines the architecture of the system and an execution σ, a path $p = [x_n : x_{n-1} : \ldots : x_0]$ between two variables x_n and x_0 defines a set of clock:*

$$\mathcal{C}lock(p).\sigma \triangleq \left\{ c_1 \circ c_2 \circ \ldots \circ c_n \;\middle|\; \begin{array}{l} \forall\, k \in [1..n], \exists\, f_k, X_{k-1} : \\ (x_k \prec f_k(X_{k-1})) \in Obs \\ \wedge\, x_{k-1} \in X_{k-1} \wedge \\ \forall\, i : x_k.\sigma_i = f_k(X_{k-1}.\sigma_{c_k(i)}) \end{array} \right\}$$

6 Timed Properties

The observation relations describe the system architecture. To complete our framework, we define the desired timed properties that specify the behavior of the variables and the relation between their timelines.

6.1 Timeline Properties

Timeline properties express the intrinsic necessity for a variable to have its value renewed often enough. That is to say, we bound the duration between two updates. In particular, this describes two behaviors: a *sporadic* variable keeps each value for a minimum duration and, on the contrary, an *alive* variable has to be updated often, no value can be kept longer than a given duration.

Definition 8 *The steadiness of a variable x is:*

$$\sigma \vDash x\{Steadiness(\delta, \Delta)\} \triangleq \forall\, i : d_x.\sigma_i \in [\delta, \Delta[$$

6.2 Relations Properties

We give timed properties on the propagation of values on a path between two variables with a set of predicates on the clocks $\mathcal{C}lock$ defined in the section 5.3

Definition 9 *Given a path $Path$ between two variables y and x, we define the parametrized relation between y and x defined by $Path$.*

$$\sigma \vDash Path \left\{ \begin{array}{c} Predicate_1(\delta_1, \Delta_1), \\ Predicate_2(\delta_2, \Delta_2) \ldots \end{array} \right\} \triangleq$$
$$\exists\, c \in \mathcal{C}lock(Path).\sigma : Predicate_1(c, \delta_1, \Delta_1) \wedge$$
$$Predicate_2(c, \delta_2, \Delta_2) \wedge$$
$$\ldots$$

$$
\begin{array}{rcl}
Lag(c,\delta,\Delta) & \triangleq & \delta \leq \hat{y}.\sigma_i - \hat{x}.\sigma_{c(i)} < \Delta \\
Latency(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_i - \hat{x}.\sigma_{c(i)} < \Delta \\
Shift(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_i - T.\sigma_{c(i)} < \Delta \\
Freshness(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_{c(i)} - \hat{x}.\sigma_{c(i)} < \Delta
\end{array}
$$

Figure 5. Predicates Characterizing The Link Between Two Variables

Such a relation is satisfied if among the clocks that bind the variables y and x, there is at least one clock that satisfies the predicates. Henceforth, for a relation between two variables y and x and a clock c, we use the predicates given in Figure 5:

- Predicate Lag is used to limit the time between the update of the source and the corresponding update of the image.
- Predicate $Latency$ quantifies the time elapsed since the appearance of the image's current value on the source and the current instant.
- Predicate $Shift$ bounds the time between the current instant and the instant when the current image value was taken on the source.
- Predicate $Freshness$ restricts the time intervals during which a source value is observable. The observation clock insures the source values are picked out during these intervals.

6.3 Example

We illustrate the timed properties on the example. The timed properties of the system are expressed on the observation relations of the system architecture definition (Figure 3) and are given Figure 6.

The properties of the variables $speed$ and $chosen$ are expressed using the steadiness. The variable $speed$ has the $Steadiness$ predicate parametrized by two bounds since it is not renewed too often in order to be significant and since it must still be updated often enough to be consistent with the real speed of the vehicle.

Due to the physical properties of the communication bus, the related observation relations are parametrized by a minimum bound on the $Shift$ predicate. These states that the value of a variable cannot be sent faster than the communication bus enables it. An upper bound is added in order to guarantee that the value available to the control system is consistent with the real value of the variables.

In order to give the timed requirements on the values used to control the vehicle speed, we express these requirements as parameters on the full processing chain. First we give requirements on the relation binding the value of variable '$actuator$ to the value of $speed$ through the system observation relations. The value of $speed$ is mostly valid when it has just been updated. We want the total time elapsed between the appearance of this value and its use as the $actuator$ value to be short so we give a $Latency$ predicate to parameter this relation. In the relation binding the '$actuator$ and the $chosen$ variables, the value of the $chosen$ variables is always "timely correct" as it may not change during a cruise. The value used to produce the '$actuator$ value must be one that was taken recently by the $chosen$ variable. That is why we use the $Shift$ parameter.

$$
\begin{array}{l}
\texttt{- variables behaviours:} \\
\qquad speed\ \{Steadiness(\delta_1,\Delta_1)\} \\
\qquad chosen\ \{Steadiness(\delta_2,+\infty)\} \\
\texttt{- communications properties:} \\
\qquad [\text{`}speed : speed]\ \{Shift(\delta_4,\Delta_4)\} \\
\qquad [\text{`}chosen : chosen]\ \{Shift(\delta_4,\Delta_4)\} \\
\qquad [\text{`}actuator : actuator]\ \{Shift(\delta_4,\Delta_4)\} \\
\texttt{- complete processing chains:} \\
\quad [\text{`}actuator : actuator : \text{`}speed : speed]\ \{Latency(0,\Delta_5)\} \\
\quad [\text{`}actuator : actuator : \text{`}chosen : chosen]\ \{Shift(0,\Delta_6)\}
\end{array}
$$

Figure 6. System Timed Properties

6.4 Specification of a System

Finally, a specification is given by a couple $\langle Archi, Prop\rangle$. $Archi$ is a set of observation relations that describes the architecture of the system. $Prop$ is a set of properties. Some are intrinsic properties that define when the variable values are renewed; some are relation properties that are parametrized by a set of predicates and that define the relation between the values of two variables. We call $SPath$ the set of paths that are used to define the timed properties of the system.

7 Feasibility Analysis

The specification defines a state transition system where the timelines of the variables are restricted by the timed properties. The system is feasible if the specification defines at least one infinite execution. We build here a transition relation that defines a system equivalent to the specification.

7.1 Definition of the Analysis System

The transition relation of the system describes the behavior of the variables of the system with respect to their relations and timed properties. A variable transition relation describes the behavior of one variable. It defines which values can be used for an update and when an update occurs. We define the global transition relation of the system as the conjunction of variable transition relations.

Definition 10 *Given the variables defined by the architecture of the system $X = \{x_k | k \in [1..n]\}$ and the corresponding variable transition relations defined by the timed properties, $\rho = \{\rightarrow_k | k \in [1..n]\}$ the global transition relation is defined by:*

$$\sigma_i \rightarrow \sigma_{i+1} \triangleq T.\sigma_{i+1} = T.\sigma_i + 1 \ \wedge \ \bigwedge_{k=1}^{n} \sigma_i \rightarrow_k \sigma_{i+1}$$

Note that each global transition induces a time step. We now explain how the transition relation of each variable is built.

7.2 Interval of Validity

The timed properties of the specification limit the instants a value can be used to produce other variables values. For each value, two time intervals are defined: the possible update instants, i.e. when a new value can be assigned to the image; and how long this value can be kept. These intervals depend on the predicates that parametrize the relation between the variables. For example, the Lag predicate defines the possible update instants of the image and the $Shift$ or $Freshness$ predicates define the instants a value can be used. Each predicate defines an interval, all predicates must be satisfied so the timed property defines an interval that is the intersection of all predicate intervals.

These intervals also depend on the timed characteristics of the value that is used. The following type is introduced to store the characteristics of a value:

$$Value = \langle \mathbb{T}, \mathbb{T}, Path \rangle$$

It stores the timed characteristics of a value of the source that is propagated along a path $Path$. The two elements in the time domain $\mathbb{T}$ are the update instant of this value and the duration this value was continuously kept. These timed characteristics define when a value can be used.

When we consider a timed property between two variables, if we know the timed characteristics of a value, then we can define the intervals of instants where this value can be used to produce a new value of the image and when this value can be kept. We define two functions that give these intervals.

Definition 11 *A time property between two variables y and x along a path p defines an interval $UpdateValid$ when a value $v = \langle \hat{x}, d_x, p \rangle$ of x which appeared at the instant $\hat{x}$ and kept for a duration of d_x can be used to update y and replace a value updated at $\hat{y}$.*

$$UpdateValid(\langle \hat{x}, d_x, p \rangle, \hat{y}) \triangleq \left[\begin{array}{c} max \left(\begin{array}{c} \hat{y} + \delta_{Steadiness}, \hat{x} + \delta_{Lag} \\ \hat{x} + \delta_{Latency}, \\ \hat{x} + \delta_{Freshness} + \delta_{Shift} \end{array} \right), \\ min \left(\begin{array}{c} \hat{y} + \Delta_{Steadiness}, \hat{x} + \Delta_{Lag}, \\ \hat{x} + \Delta_{Latency}, \\ min(\hat{x} + d_x, \hat{x} + \delta_{Freshness}) + \Delta_{Shift} \end{array} \right) \end{array} \right]$$

It also defines an interval $ValueValid$ of the instants this value can be kept.

$$ValueValid(\langle \hat{x}, d_x, p \rangle, \hat{y}) \triangleq \left[\begin{array}{c} max \left(\begin{array}{c} \hat{x} + \delta_{Latency}, \\ \hat{x} + \delta_{Freshness} + \delta_{Shift} \end{array} \right), \\ min \left(\begin{array}{c} \hat{x} + \Delta_{Latency}, \hat{y} + \Delta_{Steadiness}, \\ min(\hat{x} + d_x, \hat{x} + \delta_{Freshness}) + \Delta_{Shift} \end{array} \right) \end{array} \right]$$

7.3 The History of Values

In an observation relation $y \prec f(X)$, the value of y depends on the values of the variables of X. So when building a new value for y, we must check that the values of these variables are correct. Moreover, y can be linked to other variables through X. So we must also know which values of other variables are used to build X value.

Definition 12 *Given an execution σ, a value v contains the timed characteristics about a path p in a state σ_i if we have:*

$$\begin{array}{rl} Charac(v, i, p).\sigma \triangleq & \exists\, p', \exists\, z : p = p' :: [z] \\ & \wedge \forall\, c \in \mathcal{C}lock(p).\sigma : \\ & \quad v = \langle \hat{z}.\sigma_{c(i)}, d_z.\sigma_{c(i)}, p \rangle \end{array}$$

The operator :: defines the concatenation of two sequences. Such a value stores the timed characteristics of the value of the source of the path that is used to set the current value of the path's image. This is the value of the source in the instants pointed by the clocks of the path. For a state σ_i, such a value is unique. We create a set with the timed characteristics of the sources of the paths that are used to set the value of a variable. We are only interested in the paths used to describe the timed properties of the specification.

Definition 13 *For a variable x and an execution σ, we define the set of values that are used to build the value of x in a state σ_i and that are linked to x through a set of paths $SPath$.*

$$\begin{array}{l} SrcCharac(x, i, SPath).\sigma \triangleq \\ \{v | \exists\, p \in SPath : \exists\, p' : p = [x] :: p' \wedge Charac(v, i, p').\sigma\} \end{array}$$

The evolution of a variable is bound to the recent evolution of other variables and so to the value of the other variables in previous states. A transition relation is a predicate on a pair of states that defines the behavior of a system. In order to define the transition relation of the system defined by the specification, an auxiliary variable, called history, is introduced. We consider an observation relation $y \prec f(X)$ and one of its observation clock c so that $y.\sigma_i = f(X.\sigma_{c(i)})$. The clock c is increasing so only the values taken by X between states $\sigma_{c(i)}$ and σ_i are used to compute the next value of y. The variable history $H_{y \prec f(X)}$ stores the values that are used to build the values of X in these states.

Definition 14 *Given an observation relation $y \prec f(X)$, and the set of paths $SPath$ that are used to describe the timed properties of the system and that link y to other variables, the variable $H_{y \prec f(X)}$ is defined by:*

$$\begin{array}{l} \forall\, \sigma, \forall\, i : H_{y \prec f(X)}.\sigma_i \triangleq \\ \left\{ \bigcup_{x \in X} SrcCharac(x, j, SPath').\sigma \;\middle|\; \begin{array}{l} c \in \mathcal{C}lock([y : x]).\sigma \\ \wedge\, x \in X \;\wedge\; j \in [c(i)..i] \end{array} \right\} \end{array}$$

where

$$SPath' = \{[x] : p \mid x \in X \;\wedge\; ([y : x] :: p) \in SPath\}$$

History variable is a set of sets of values that gives the characteristics of the values that can be used to build the next value of the image. The values that are linked to the same value of X in the same instant are stored in the same set. A partial order relation that is based on the chronological order is defined on the set of values that are stored in the history.

Definition 15 *An order relation is defined between the set of values and for an execution σ.*

$$\forall\, \mathcal{V}_1, \mathcal{V}_2 : \mathcal{V}_1 < \mathcal{V}_2 \triangleq \begin{array}{l} \exists\, i, j : i < j \ \wedge\ \exists\, x, \exists\, SPath : \\ \mathcal{V}_1 = SrcCharac(v_1, i, SPath).\sigma \\ \wedge\, \mathcal{V}_2 = SrcCharac(v_2, j, SPath).\sigma \end{array}$$

7.4 Variable Transition Relation

We define here the transition relation for a variable y image of an observation $y \prec f(X)$ that describes the evolution of y when time is increased.

There are two possible evolutions: the image is updated with a new value or the same value is kept. The possibility to use a value does not only depends on the value taken by the variables in X. It also depends on the values used to produce X value. Given the architecture of the system, we check that all values used to produce y value satisfy the specification. So we check the characteristics of the values in the history. The intervals defined in section 7.2 are used to define the variable transition relation.

Definition 16 *For each variable relation $y \prec f(X)$ in a specification, a variable transition relation is defined by:*

$$\forall\, \sigma_1, \sigma_2 : \sigma_1 \rightarrow \sigma_2 \triangleq \left(\begin{array}{c} \left(\begin{array}{c} \hat{y}.\sigma_2 \neq T.\sigma_2 \ \wedge\ \forall\, v \in min(H_{y \prec f(X)}.\sigma_2) : \\ T.\sigma_2 \in ValueValid(v, \hat{y}.\sigma_1) \end{array} \right) \\ \vee \\ \left(\begin{array}{c} \hat{y}.\sigma_2 = T.\sigma_2 \ \wedge\ \forall\, v \in min(H_{y \prec f(X)}.\sigma_2) : \\ T.\sigma_2 \in UpdateValid(v, \hat{y}.\sigma_1) \end{array} \right) \end{array} \right)$$

The history only stores the values of the sources no older than the values that set the image current value (pointed by c). So $min(H_{y \prec f(X)})$ denotes the set of values currently used to define the image value. The state σ_2 is influenced by the state σ_1 through the definition of the history and the instant when the value of y in σ_1 was updated.

For a variable that is not the image of an observation relation, the dedicated transition relation is only defined by its intrinsic timeline property.

7.5 Reduction and Exploration of the System

We proceed to the analysis of the system defined by the global transition relation. We must explore the executions to prove the existence of an infinite execution and thus to prove the system feasibility. However the specification defines a system with an infinite number of states. There is no bound on the time T and other timed variables such as the update instants. So these variables can take an infinite number of values.

In order to perform a finite exploration of the states of an execution, we build a system bi-similar to the specification but where the variables take a finite number of values. This is possible if the shift between all timed variables is bounded. So this is possible if the specification states upper bounds on the time a value can be used by the system.

Given an observation $y \prec f(X)$, we bound the values of the timed characteristics that are used to check the validity of the value assigned to y. We look for a bound on all update instant stored in the history variable. All these values are in the interval between the update instants of the values at the beginning of the paths and the current time.

Proposition 1 *Given an observation $y \prec f(X)$ we have:*

$$\forall\, i : \forall\, \mathcal{V} \in H_{y \prec X}.\sigma_i, \forall\, \langle v_{\hat{x}}, v_{d_x}, p \rangle \in \mathcal{V} : v_{\hat{x}} \in [min_{src}, T.\sigma_i, v)]$$

where:

$$min_{src} = min\left(\left\{ \hat{s}.\sigma_{c(i)} \;\middle|\; \begin{array}{c} \exists\, p \in SPath, \exists\, p' : \\ p = [y] :: p' :: [s] \wedge \\ c \in \mathcal{C}lock(p).\sigma \end{array} \right\} \right)$$

We want to give a constant maximum size of this interval in all states. In a system where the relations between the variables and the sources are parametrized by a $Latency$ predicate, then the shift between all variables is bounded in all states by the most permissive $Latency$ predicate i.e. the one with the maximum upper bound. We restrict our analysis to such systems. If this property is not explicitly stated in the specification, then we use a set of propositions. Here are their principles:

- the latency that parameters an observation relation can be deduced from the combination of other predicates that parameters the relation such as $Steadiness$ and Lag;
- if along a path defined by a set of observations, all observations are parametrized by a latency predicate then so is the full path;
- if there are multiple sources all with a $Steadiness$ predicate parametrizing their behavior, and if there is a $Latency$ predicate binding one of this source to the image, then all are bound to the image with a $Latency$ predicate.

In the example, there is no upper bound on the $Steadiness$ predicate of the variable $chosen$. For the system to be analyzable, such a property must be added. A large bound must be chosen in order to not interfere with other properties.

Based on these properties and for such a system, we define a system where all values of the instants are stored modulo the length of an analysis interval denoted by L. L is chosen by the specification as a bound greater than the upper bounds on the variables $Steadiness$ and the paths $Latency$.

In the system defined by the specification, transitions are based on the time differences between the instants characterizing the variable timelines. These differences do not exceed the length L. Thus, for each state, if the value

of the time T is known and if the values of the other variables are known modulo L, then for each variable there is only one possible real value that can be computed using the value of T. Considering the clock values modulo this length does not add or remove any behavior of the original system.

Consequently the equivalence relation that is defined by the equality of timed variables modulo L is a bi-simulation. In this system, all timed variables have a finite number of values. So we obtain a system that is bi-similar to the system defined by the specification and that has a finite number of states.

In order to prove the system feasibility, we explore the executions that are defined by the finite system using a depth first search algorithm. A loop denotes an infinite execution. This proves the feasibility of the specification.

8 Verification of an Implementation

We explain here how to check an implementation. An implementation is correct if each of the behaviors it defines satisfy the specification. In other words, the implementation defines a set of executions that must be included in the executions defined by the specification.

8.1 Value Availability

In order to analyze an implementation, it must be modelled. An implementation defines how the values taken by each variable are transmitted through the architecture of the system. To check the timed properties of the implementation, we focus on the implementation properties that define the instants when a source value is available to the image.

Given an observation relation $y \prec f(X)$ and for each value taken by X, we define different states of availability. These states are the different steps between the instant when a value is assigned to the source and the instant when the image value is bound to this source value:

- initial (It): the value is currently assigned to the source;
- sent (St): the value has been stored to be later available to the image. For example a message has been created containing the source value or a component has read the inputs used for the computation;
- received (Rd): the value is available to the image. The message containing the value of the source has been received or the computation of the image new value is completed;
- delivered (Dd): the value of the source has been used to set the current value of the image.

Each value is in one and only one of the sent, received or delivered states but it may be both in the initial state and in another state. These states of availability do not exactly describe the different states of a value in an implementation. But the behavior of an implementation can be modelled with these states.

The history variable is split into different sets of values depending on the availability of each value. These sets are in fact sequence of sets of values. They are ordered with the order relation on sets of values (chronological order, the oldest one is the first in the sequence).

Definition 17 *Given an observation relation $y \prec f(X)$ and the paths $SPath$ used to describe the timed properties of y, the sequences that describe the states of availability of the values are defined by:*

$$\begin{array}{l}
\forall\, \sigma, \forall\, i : \\
It.\sigma_i \triangleq \{ \bigcup\limits_{x \in X} SrcCharac(x, i, SPath).\sigma \} \\
\wedge\ Dd.\sigma_i \triangleq \left\{ \begin{array}{r} \bigcup\limits_{x \in X} SrcCharac(x, c(i), SPath).\sigma \\ \mid c \in \mathcal{C}lock([y : x]).\sigma \end{array} \right\} \\
\wedge\ (It :: St :: Rd :: Dd).\sigma_i \subseteq H_{y \prec f(X)}.\sigma_i
\end{array}$$

The sequences It and Dd are singletons. A value goes through the four states chronologically.

8.2 Model of the Specification

In order to check the satisfaction of the specification by an implementation, we give a model of the specification in the same semantic we use to model an implementation. Such a model is described by defining elementary transitions. An elementary transition relation model the evolution of the values states of availability in the observation relations of the system. These elementary transition relations are used to build the variable transition relation of the image of an observation. The variable transition relations are then used to build the global transition relation.

We use a semantic close to TLA [5] that is based on actions. An action is a predicate on two states, and an elementary transition relation is defined as a disjunction of actions. The actions give the different evolutions of the availability of a value and when these evolutions are allowed by the specification.

Definition 18 *Given a set of the actions $A = \{a_k | k \in [1..n]\}$ for the evolution of a value from one state of availability to the next one, we define an elementary transition relation $\rightarrow$*

$$\forall \sigma_i, \sigma_j : \sigma_i \rightarrow \sigma_j \triangleq \bigvee_{k=1}^{n} a_k.\sigma_i.\sigma_j$$

We now define the actions used to build a model of the specification. Except if it is stated by the action, all variables have the same value in both states of an action.

8.2.1 Sender

This elementary transition relation rules the evolution of the current value of the image to the sequence St. There are two actions: the value can be sent or not.

$$\begin{array}{l}
\forall \sigma_i, \sigma_j : \\
\quad Send.\sigma_i.\sigma_j \triangleq St.\sigma_j = St.\sigma_i :: It.\sigma_j \\
\quad Idle.\sigma_i.\sigma_j \ \triangleq true
\end{array}$$

The current value is sent when the value in the sequence It is added to the sequence St.

8.2.2 Receiver

This elementary transition relation rules the evolution of the values from the sequence St to the sequence Rd. Each value can be passed to the next sequence or lost.

$$\begin{array}{l}\forall \sigma_i, \sigma_j : \\ \quad Rcv(St_L, St_R).\sigma_i.\sigma_j \triangleq St.\sigma_i = Merge(St_L, St_R) :: St.\sigma_j \\ \qquad\qquad\qquad\qquad \wedge\ Rd.\sigma_j = Rd.\sigma_i :: St_R \\ \quad Lose(St_L).\sigma_i.\sigma_j \quad \triangleq St.\sigma_j = St.\sigma_i \backslash St_L\end{array}$$

The function $Merge$ builds the ordered sequence union of two sequences. The actions Rcv passes the values St_R from St to Rd but lose the values in St_L. The actions are parametrized by sequence of values since the possible actions depend on the number of values in the sequence St.

8.2.3 Image

This relation decides which value is assigned to the image. It can keep the same value or take a new value that is in the sequence Rd. Some values of the sequence Rd can also be removed from this sequence. For an observation $y \prec f(X)$ we have:

$$\begin{array}{l}\forall \sigma_i, \sigma_j : \\ \quad Update(\mathcal{V}, Rd_L).\sigma_i.\sigma_j \triangleq \hat{y}.\sigma_j = T.\sigma_j \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Rd.\sigma_i = Merge([\mathcal{V}], Rd_L) :: Rd.\sigma_j \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Dd.\sigma_j = [\mathcal{V}] \\ \qquad\qquad\qquad\qquad\qquad \bigwedge_{v \in \mathcal{V}} T.\sigma_j \in UpdateValid(v, \hat{y}.\sigma_i) \\ \quad Keep(Rd_L).\sigma_i.\sigma_j \quad \triangleq Rd.\sigma_i = Rd_L :: Rd.\sigma_j \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Dd.\sigma_i = [\mathcal{V}] \\ \qquad\qquad\qquad\qquad\qquad \bigwedge_{v \in \mathcal{V}} T.\sigma_j \in ValueValid(v, \hat{y}.\sigma_i)\end{array}$$

The action $Keep$ check that the values can be kept by using the predicate $ValueValid$. The action $Update$ is defined with the predicate $UpdateValid$ that checks that y can be updated with the new value.

8.2.4 Variable Transition Relation

We build the variable transition relation of a variable y by using the elementary transition relation of the observation which image is y. The variable transition relation is defined as a sequence of elementary transition relations.

Definition 19 *Given an observation $y \prec x$ and the elementary transition relations:*

$$\rightarrow_{sd}; \rightarrow_{rcv}; \rightarrow_{img}$$

Then the transition relation $\rightarrow_y$ that defines the behavior of y is:

$$\begin{array}{l}\forall \sigma_i : \sigma_i \rightarrow_y \sigma_{i+1} \triangleq \\ \qquad \exists\, \sigma_s, \sigma_r : \sigma_i \rightarrow_{sd} \sigma_s \rightarrow_{rcv} \sigma_r \rightarrow_{img} \sigma_{i+1}\end{array}$$

The intermediary states between the elementary transitions are hidden to ensure a separated execution. A definition of the variable transition relation could be given as a conjunction of elementary relations but this definition eases the expression of the elementary relation transitions. The global transition relation is defined as the conjunction of the variable transition relations as we did in the feasibility analysis.

This model of the specification is equivalent to the state transition system defined for the feasibility analysis. The specification do not allow loss of values, but a loss is equivalent to not finally using this value to update the image. Therefore this model is bi-similar to the specification.

8.3 Model of the Implementation

The implementation is modelled by redefining the same actions as the specification. So we describe the evolution of the values through the same states of availability and use the same four sequences. We use some part of the example to illustrate how to define such a model. We first discuss the properties of the communication protocol. We suppose all communications are done through the same communication bus. To model a communication protocol, two characteristics need to be abstracted: when are the messages sent and what is the communication time. Moreover, are these characteristics determinate or is there a jitter? In a time triggered protocol, the evolution to the sent availability state is decided by the value of the time T. The evolution to the received state depends on the communication time. The value in the availability sequences are redefined as a new type that contains the instant when a value is sent.

$$Value = \langle \mathbb{T} \rangle$$

If the messages are sent with a period of P with a phase ϕ and if the communication time is d then we define the following actions for a communication such as '$speed \prec speed$. In the example no loss is allowed. Only one value at a time can be received, and all values that are received are directly assigned to the image.

$$\begin{array}{l}\forall \sigma_i, \sigma_j : \\ \quad Send.\sigma_i.\sigma_j \quad \triangleq T.\sigma_j = \phi \pmod P \\ \qquad\qquad\qquad\qquad \wedge\ It.\sigma_j = \{\langle T.\sigma_j \rangle\} \\ \quad Idle.\sigma_i.\sigma_j \quad \triangleq T.\sigma_j \neq \phi \pmod P \\ \quad Rcv(St_L, St_R).\sigma_i.\sigma_j \quad \triangleq St_R = \{\langle T_{sent} \rangle\} \\ \qquad\qquad\qquad\qquad\qquad \wedge\ T.\sigma_j - T_{sent} = d \\ \qquad\qquad\qquad\qquad\qquad \wedge\ St.\sigma_i = St_R :: St.\sigma_j \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Rd.\sigma_j = Rd.\sigma_i :: St_R \\ \qquad\qquad\qquad\qquad\qquad \wedge\ St_L = \emptyset \\ \quad Lose(St_L).\sigma_i.\sigma_j \quad \triangleq St_L = \emptyset \\ \quad Update(\mathcal{V}, Rd_L).\sigma_i.\sigma_j \triangleq Rd.\sigma_i = [\mathcal{V}] \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Rd.\sigma_j = \emptyset \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Dd.\sigma_i = [\mathcal{V}] \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Rd_L = \emptyset \\ \quad Keep(Rd_L).\sigma_i.\sigma_j \quad \triangleq Rd_L = \emptyset \\ \qquad\qquad\qquad\qquad\qquad \wedge\ Rd.\sigma_i = \emptyset\end{array}$$

Here the period P is used to define when the $Send$ action is allowed and so when values are passed to the sent availability state. The communication time and the instant a value is sent are used in the Rcv action. They define when the values are passed to the received availability state.

In an observation that models a computation, the availability state depends on the same kind of characteristics which are when the computation starts and the possible

execution time. The results of a scheduling analysis can be used to give these characteristics.

8.4 Correctness of an Implementation

An implementation is correct if the set of executions it defines is included in the executions that are defined by the specification. So the model of the specification must simulate the implementation. In order to check this property, we build a state transition system similar to the synchronized product of labelled transition systems. The actions are used as labels on the transition of the systems.

Definition 20 *Given the set of actions* $A_I = \{a_{I_k} | k \in [1..n]\}$ *that defines an elementary transition of the implementation and the set of actions* $A_S = \{a_{S_k} | k \in [1..n]\}$ *that defines the corresponding elementary transition of the specification, we define the set of couples of actions where the actions with the same label are joined.*

$$A = \{\langle a_{I_k}, a_{S_k}\rangle | k \in [1..n]\}$$

An elementary transition relation of the system that checks the correctness of the implementation is defined by:

$$\forall\, \sigma_i, \sigma_j : \sigma_i \rightarrow \sigma_j \triangleq (\bigvee_{k=1}^{n} a_{I_k}.\sigma_i.\sigma_j \wedge a_{S_k}.\sigma_i.\sigma_j) \\ \wedge \forall\, k, \forall\, \sigma_l : (a_{I_k}.\sigma_i.\sigma_l \Rightarrow a_{S_k}.\sigma_i.\sigma_l)$$

In this definition, the second part states that if a transition between two states is allowed by the implementation, it must be allowed by the specification. So if there is a conflict between the specification and the implementation then there is a deadlock. The global system is built by using these elementary transition relation to build the variables transition relations. Note that two actions are different if their parameters are different. For example $Rcv(\emptyset, \mathcal{V})$ is different from $Rcv(\emptyset, \emptyset)$. If the sizes of the sequences are bounded, then the number of different actions is also bounded. The sizes of the sequences are bounded if the system can be reduced to a finite system.

8.5 Reduction of the System

In order to proceed to the analysis of the system, we here also build a finite system equivalent to the system that is defined by the specification and the implementation. This is only possible if the variables introduced to define the implementation properties have properties that allow the reduction technique to be used. So the timed variables introduce by the model of the implementation must have a bounded shift to the time T. These properties are required to proceed to an automatic verification and must be stated by the user.

In this system, a deadlock exists if no behavior can be taken. A deadlock denotes either an incompatibility between the specification and the implementation or that the specification is not feasible. So the correctness of the implementation is checked with a model checking algorithm used to detect deadlocks.

9 Conclusion

We specify an abstract model postponing task and communication scheduling to specify real-time systems. Our framework proposes to specify real-time systems as a set of links between system variables. The timed properties of the system characterize the time-shift along the propagation of values in the system. They state that all the values that are used must be timely correct with respect to the user requirements. A dedicated state transition system that is bi-similar to the specification, is built to proceed to a feasibility analysis. We finally describe how to model an implementation of the system. A dedicated state transition system is also built to check the correctness of the implementation with respect to the specification. Perspectives are to enhance the implementation of the tool used to build the analysis state transition systems. This tool can then be used to proceed to the analysis of a larger scale example.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

[2] S. Anderson and J. K. Filipe. Guaranteeing temporal validity with a real-time logic of knowledge. In *ICDCSW '03: Proc. of the 23rd Int'l Conf. on Distributed Computing Systems*, pages 178–183, 2003.

[3] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Data consistency in hard real-time systems. Technical report, 1993.

[4] M. Charpentier, M. Filali, P. Mauran, G. Padiou, and P. Quinnec. The observation : an abstract communication mechanism. *Parallel Processing Letters*, 9(3):437–450, 1999.

[5] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[6] E. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[7] G. Roşu and S. Bensalem. Allen Linear (Interval) Temporal Logic – Translation to LTL and Monitor Synthesis. In *International Conference on Computer-Aided Verification (CAV'06)*, number 4144 in Lecture Notes in Computer Science, pages 263–277. Springer Verlag, 2006.

[8] X. C. Song and J. W. Liu. Maintaining temporal consistency: Pessimistic vs. optimistic concurrency control. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):786–796, 1995.

[9] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40(2-3):117–134, 1994.

[10] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling transactions with temporal constraints: exploiting data semantics. In *RTSS '96: Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 240–253, 1996.